What is Software Architecture to Practitioners: A Survey

Mert Ozkaya
Department of Computer Engineering, Istanbul Kemerburgaz University, Istanbul, Turkey

Keywords: Software Architectures, Survey, Architecture Description Languages, UML.

Abstract: Software architecture has been proposed in the nineties as a high-level software design method for specifying software systems in terms of components and their relation. Since then, software architectures have become an indispensable part of software design. However, it remains dubious to what extent practitioners use software architectures in their software design. To better understand this, we conduct a survey study among a number of practitioners from both industry and academia and aim at understanding their level of knowledge and experience in software architectures. Our survey consists of a questionnaire of 20 questions, presented in four distinct sections. We run our survey on 50 participants, 11 of whom are from academia and the rest 39 are from industry. As a result of our analysis, we reached the following conclusion: while software architecture is highly crucial for practitioners given the nature of their software projects, practitioners' knowledge on software architectures is too limited. Practitioners use Unified Modelling Language (UML), which views software architectures as a method of communicating system structures. However, other aspects such as architectural analysis are equally crucial in detecting design errors and verifying software designs for quality properties.

1 INTRODUCTION

Software architecture (Perry and Wolf, 1992; Garlan and Shaw, 1994; Clements et al., 2003) is a high-level design activity, concerned with the successful composition of components into an entire system that meets functional and non-functional requirements. It is at the level of architectural design where low-level details of components are suppressed, and, their high-level complex interactions via the component interfaces (i.e., the protocols of interactions) can be focused on and reasoned about. So, design problems, e.g., the use of interface services in the wrong order, can be identified early on at the stage of high-level design. Indeed, problems due to incompatible interfaces of inter-connected components are crucial, which prevent the components from being composed to a whole system and analysed for non-functional properties, e.g., reliability and security.

Unified Modelling Language (UML) (Rumbaugh et al., 1999) is the de facto language for visually specifying and designing software systems. UML supports both high-level and low-level designs, which is widely used in specifying high-level software architectures too. It offers a variety of diagrams, such as class and component diagrams. Using these diagrams, systems can be specified as a composition of components that are connected with each other via association links (Ivers et al., 2004). However, UML does not provide direct support for the first-class specification of interaction protocols for the linked components, which are crucial for reasoning about their composition. Moreover, UML originally has very weak formal semantics, which are open to different interpretations and not easily formally analysed.

To specify software architectures precisely and formally analyse their behaviours, designers can use process algebraic formal methods, e.g., FSP (Magee et al., 1997), CSP (Hoare, 1978), and π-calculus (Milner et al., 1992), or other formalisms, e.g., Z notation (Spivey, 1992), which are supported by analysis tools. Using these tools, system behaviours can be analysed exhaustively to detect safety issues, such as deadlock, which prevent successful composition of components. However, these formal methods have mathematical notations that are based on mathematical proofs. So, their notations are considerably different from the notations of widely used modelling languages (e.g., UML), and, thus, practitioners are likely to find formal methods unfamiliar.

Another alternative method for specifying software architectures is the architecture description languages (ADLs), which have emerged in the nineties and become one of most active areas of software engineering (Vestal, 1993; Clements, 1996; Medvidovic and Taylor, 2000; Fuxman, 2000; Woods and Hilliard,

Ozkaya, M.
What is Software Architecture to Practitioners: A Survey.
DOI: 10.5220/0005826006770686
In *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2016)*, pages 677-686
ISBN: 978-989-758-168-7

2005). There are numerous ADLs developed so far, e.g., Darwin (Magee and Kramer, 1996), UniCon (Shaw et al., 1995), Wright (Allen and Garlan, 1997), Rapide (Luckham, 1996), C2 (Taylor et al., 1996), LEDA (Canal et al., 1999), AADL (Feiler et al., 2006), SOFA (Plasil and Visnovsky, 2002), RADL (Reussner et al., 2003), etc. Each ADL offers its own architectural notation, but, they share basic notions, e.g., components, interfaces, and connectors. Unlike UML, ADLs allow designers to specify the architectures of their systems precisely. Moreover, ADLs are offered with various features depending on their scope of interest. Some offer automatic code generation for facilitating the implementation of the specified systems. Some offer notations for specifying non-functional properties of systems (e.g., reliability and security), which can be communicated among stakeholders and analysed via analysis tools. Some offer notations based on formal methods (e.g., process algebras (Bergstra, 2001)) for specifying the behaviours of architectural elements and formally verifying them using formal analysis tools, e.g., model checkers.

1.1 Survey on Software Architectures

As introduced above, there are many techniques for specifying software architectures, and, each technique has its own advantages and disadvantages. UML for instance are found easy to learn and use by practitioners thanks to its visual notation set. However, UML originally does not have formally defined semantics, which may lead to imprecise specifications that are interpreted differently. ADLs differ from UML by their precise notations, which also let designers perform further operations on their software architectures such as automatic code generation, simulation, and formal analysis. However, ADLs are not as widely-used as UML by practitioners, since ADLs are based on formal methods to enable formal analysis that make their learning curve steep too.

Given these software architecture design techniques discussed above, one would hope that software architecture must have already entered the mainstream of practitioners and used by them as a high-level design activity. However, it is not yet exactly known whether practitioners are aware of these techniques and use them in their software architecture specifications. To better understand this, we conduct a survey among a number of practitioners. Our main focus is to find answers for the following research questions:

- what do practitioners understand from software architectures?
- what do practitioners aim at achieving by specifying software architectures and are they aware of facilities such as architectural analysis?

We run a survey among 50 participants: 39 of them are industrial experts and 11 of them are academics. Our survey consists of 20 different questions, divided up into 4 different sections. Each section includes a set of questions for a particular concern, which are given below.

1. The types of the projects conducted by practitioners
2. Practitioners' understanding of software architectures
3. Practitioners' interest towards the benefits of specifying software architectures
4. Software architecture specification techniques used by practitioners

The first survey section above aids in exploring whether practitioners develop such software systems in which specifying the software architectures brings potential benefits to the development. The second section explores what software architecture means to practitioners. By this, we particularly aim at understanding practitioners' level of knowledge on software architectures. The third section explores practitioners' level of interest towards software architectures. More specifically, we attempt at understanding here whether practitioners utilize from their software architecture specifications, e.g., analysing them for quality properties (e.g., reliability and performance) and generating implementation code from them. The last section explores the particular techniques that practitioners prefer to use for specifying software architectures.

2 RESEARCH METHODOLOGY

We run the survey in three subsequent stages. First, we planned our survey, deciding on the types of participants and the methods for participation. Second, we designed our survey, deciding on the questions to be asked to the participants. Lastly, we worked out the analysis methods to be applied in analysing the gathered survey data.

2.1 Planning our Survey

Our first step was to identify the group of practitioners whom we will contact for requesting to fill in our survey. We focussed on two groups: industrial experts and academics who conduct research and development in software engineering.In the rest of this section, we introduce our survey plan, depicted in Figure 1.

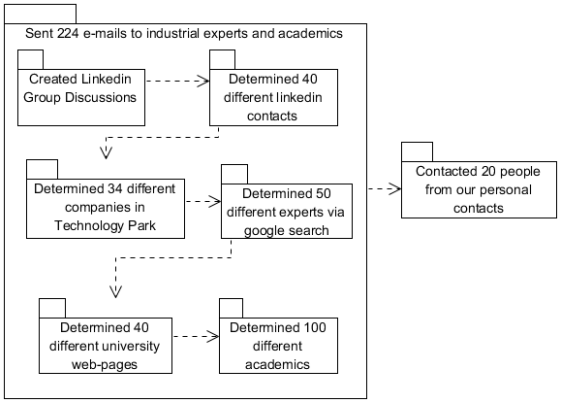

Figure 1: Survey Plan.

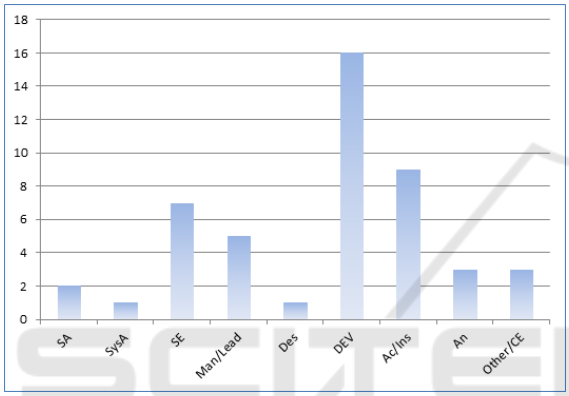

Figure 2: Participant roles in industry.

SA:Software Architect SysA: System Arc. SE: Soft Eng Man/lead: Manager leader Des: Designer Dev: Developer Ac/Ins: Academics and instructors in universities An: Analyst Other/Ce: Computer engineer

Industrial Experts. We contacted a number of experts in software industry via the following methods and requested them to fill in our survey. The role of the industrial experts who have participated our survey are depicted in in Figure 2.

We initially used social media (e.g., linkedin) so as to reach groups of relevant people. We determined 40 different contacts and got in touch with them via linkedin's e-mail service or by company phone numbers (if available). We also went through the web-sites of the technology parks that operate in Turkey and identified the e-mail addresses or telephone contacts of the software development companies based in the technology parks. By doing so, we got in touch with 34 different companies. Lastly, we searched in google for the software development companies that we have not encountered during our linkedin and technology park searches. By doing so, we determined 50 different experts and contacted them either by e-mail or phone-call.

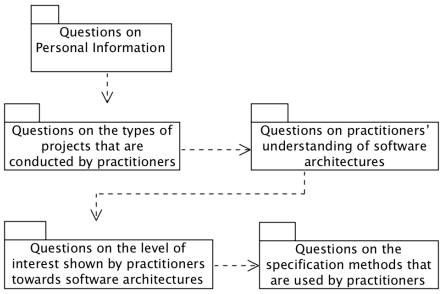

Figure 3: Survey Questions.

Academics. Besides industrial experts, we also targeted at the academics who work for the computer engineering (or computer science) department of universities. We identified the academics and their contact details by browsing the university web-pages.

We browsed through 40 different universities in Turkey and determined the e-mail addresses of the academics from their personal web-pages. In total, we sent e-mails to 100 different academics for requesting their participation.

In conclusion, we sent 224 e-mails to industrial experts and academics in total and got in touch with many others indirectly via group announcements. Currently, we have got 48 replies: 10 of them told that they cannot fill in our survey due to their company policy. The rest 38 accepted to fill in our survey. Moreover, we also used our personal contacts and contacted 20 people consisting of industrial experts and academics. We have got 12 replies, who accepted to fill in our survey.

2.2 Designing our Survey Questions

In this survey project, we aim at exploring to what extent practitioners know about software architecture and apply it in their software design. Therefore, we structured and designed our survey questions in a way that aids in understanding practitioners' knowledge and experience about software architectures. As depicted in figure 3, we proposed five different sections in our survey, each consisting of a number of questions. Below, we discuss these survey sections and the questions in each section.

2.2.1 Personal Information

This section is for gaining some personal information about the participants so as to argue their answers to the survey questions in a more sensible way. We ask three questions in this section and try to learn *(i)* whether the participant is an industrial expert or an academic, *(ii)* the participant's job position, and *(iii)*

the company or institution for which the participant works.

2.2.2 The Kinds of Software Systems that are Developed by Practitioners

In this section, we aim at understanding whether practitioners are involved in the development of software systems in which software architectural design could facilitate the development. Initially, we try to learn whether the participant has ever built a critical software system whose failure may lead to catastrophic results. As mentioned in Section 1, through ADLs, it is possible to specify the architecture of such critical systems and analyse their behaviours for quality issues. Second, we ask to learn whether the participant works in projects with multiple development teams that can work concurrently. If so, specifying software architectures will let teams work concurrently on separate independent system components. Third, we ask to learn whether the participant builds long-lived projects that require constant maintenance effort. Specifying the architecture of systems in terms of independent components will be very useful as one can more easily understand the system functionalities and modify them when needed. Lastly, we ask whether the participants build families of systems, such as client-server systems, which can be designed effectively by using architectural styles.

2.2.3 Practitioners' Understanding of Software Architectures

In this section, we seek to understand how well participants know about software architectures. We firstly ask to learn what the participants think that software architects do. The second question here is for learning the participants' thoughts on possible reasons for specifying software architectures. In the last three questions, we learn the participants' research or teaching background on software architectures.

2.2.4 The Level of Interest Shown by Practitioners towards Software Architectures

Besides understanding the participants' level of knowledge on software architectures, we also wish to understand the participants' level of interest on software architectures. Firstly, we ask to learn if the participants always consider the architecture specification of his/her software system while designing their system. Secondly, we ask for the ADLs that the participant is aware of (if any). Next, we ask when the participants prefer to check the correctness of their software and whether they know that architectural design of software systems can be verified for correctness using architecture description languages. Lastly, we ask the participants which method they follow in implementing their software system and whether they know that architectural designs can be translated into implementation code by using architecture description languages.

2.2.5 The Specification Methods that are used by Practitioners

In this last section, we ask three different questions to learn the tools/languages that the participants use for specifying software architectures. Firstly, we ask the participants which tool/language they use (if any). Next, we ask them to state their observations on the advantages and disadvantages of the tool/language (if any).

2.3 Analysing Survey Responses

In this part, we describe how we aim at analysing the data collected through the survey.

Survey Data Collection and Organisation. We conduct our survey online via Google Forms. Google Forms produces an Excel file, which is updated automatically upon each participant filling the survey. The excel file includes a unique column for each survey question; and, a new row is added automatically when a participant answered the survey questions.

Data Analysis. We aim at analysing the collected (and organized) data in three ways.

Calculating the Response Rate: Firstly, we calculate the response rate, which is the percentage of the received response after contacting a particular number of participants. By doing so, we aim at determining the level of interest shown towards our survey and taking some actions for improving the response rate, especially if it is considerably low.

Calculating the Response Frequencies: For each different section in the survey, we go through its questions one by one. We aim at analysing the responses given to each question by calculating the percentage of each answer. We create a table per question, which consists of two rows and a single column. While in the first row we give the question itself, in the second row we provide the respective analysis data for that question obtained from the survey (i.e., the percentage of each possible answer for the question given by the participants).

Computing the Cross-tabulations: Besides analysing the questions individually, we are also interested in revealing the relationships between

different questions (if any). Indeed, for some questions, the answers given to them may be understood better and lead to more interesting conclusions if we consider those answers together with the answer(s) given by the same participants to another question(s). For instance, while we wish to know the percentage of the participants who use UML, we are more interested in the participants who use UML and at the same time look for easy to learn and use languages.

Coding: In our survey, some questions allow participants to type their own responses as an alternative to choosing from a pre-defined list of responses. For instance, when asking what software architects would do in a software company, we also offer participants the choice of writing their own view. However, participants may choose to write their own view although their view is very similar (if not the same) to one of the pre-defined choices. Therefore, we apply the coding technique here, and, go through the responses typed by the participants and re-format (and sometimes re-write) their answers so as to match them with the closest pre-defined choices.

3 MAIN FINDINGS AND RESULTS

3.1 Practitioners Can Facilitate the Development of their Systems through Software Architectures

In Section 2.2.2, we introduced a set of questions for understanding what kind of software systems that the participants develop.

Table 1 in the Appendix page 9 gives the response frequencies for each of the questions. According to our survey results, 80% of the participants have responded "Yes" to at least one of the questions. That is, every 8 participants out of 10 develop either *(i)* critical systems, *(ii)* large systems with more than one development teams, *(iii)* long-lived systems, or *(iv)* families of systems/products. As already discussed in Section 2.2.2, specifying the software architectures of such systems could facilitate their development and aid in detecting any design errors before they propagate them in the implementation.

3.2 The Analysis of Software Architectures is a New Concept to Participants

In Section 2.2.3, we introduced the part of the survey that includes a set of questions for determining how well the participants know software architectures.

Table 2 in the Appendix page 9 gives the response frequencies for each of the questions. Apparently, 77% of the participants have already got introduced with software architectures, who have either taken/given courses on software architectures, attended seminars/conferences, or done relevant researches. However, many participants are still unfamiliar with the role of software architects, believing that software architects just design the structure of the system. They miss the point that one of the major tasks that can do with software architectures is the analysis of software architectures for quality issues, e.g., reliability and performance. 87% of the participants who have taken/given courses on software architectures failed to choose the most complete answer for Q1, i.e., "A software architect decomposes a system into components, and then analyse these components along with their interaction for quality properties". This reveals that practitioners view software architectures as a high-level design method for specifying software systems in terms of components and their relation. Their main intention in specifying software architectures is to facilitate the understanding and communication of large and complex software systems by dividing them into smaller parts (i.e., components). The ability of analysing software architectures for detecting design errors and quality issues (e.g., performance and security) is ignored. Even 60% of the participants with the academic role are not aware of analysing software architectures - they did not provide the correct answer for Q1. So, this leads to their students (or potential practitioners) who have a narrow scope on software architectures.

3.3 Architecture Description Languages (ADLs) are Shown Lack of Interest by the Participants

In Section 2.2.4, we introduced the part of the survey that includes a set of questions for understanding the level of interest shown by the participants towards architecture description languages.

Table 3 in the Appendix page 9 gives the response frequencies for each of these questions. 68% of the participants who consider software architectures in their software design are not aware of any ADLs. This

means that most practitioners specify their software architectures either using UML or box-line diagram drawing tools. However, neither UML nor box-line drawings helps in analysing software architectures for design errors and quality properties[1]. Considering that 85% of those participants are interested in checking software correctness at design time, this is quite concerning. Even among the participants both considering software architectures and aware of architecture description languages, some still do not know of the analysis capabilities of architecture description languages (33%). This is probably because they heard about ADLs but have not used them for analysis purposes.

According to the survey results, practitioners are also interested in implementing their software systems in accordance with their software design (57%). This is indeed facilitated through ADLs, which offer automatic code generation from software architectures. However, 60% of the participants who prefer to implement their software design are unaware of the code generation facilities of ADLs.

3.4 UML is Popular with its Low Learning Curve and Visual Notations, while Still Suffering from the Lack of Support for Architectural Analysis

In Section 2.2.5, we introduced the part of the survey that includes a set of questions for understanding the techniques used by the participants for specifying ADLs.

Table 4 in the Appendix page 9 gives the response frequencies for each of the questions. According to the survey results, 89% of the participants use UML for specifying software architectures. 63% of the participants who use UML look for software design techniques that are easy to learn and use. Likewise, 75% of the participants who use UML look for visual notations. This shows that UML is found popular thanks to its low learning curve and comprehensive set of visual notations. Moreover, 41% of the participants who use UML wish that architectural analysis was supported, which UML lacks in unfortunately.

[1]Although there are some attempts towards extending UML through UML profiles so as to support architectural concepts (e.g., connectors) and analysis, UML does not originally provide direct support for connectors and analysis of software architectures.

4 DISCUSSION

In this survey, we asked 20 different questions to 50 participants with the goal of finding answers for the following two research questions: *(i)* what do practitioners understand from software architectures? *(ii)* what do practitioners aim at achieving by specifying software architectures and are they aware of facilities such as architectural analysis? Having discussed the answers given to the survey questions in Section 3, we observed that architectural design can be very helpful for practitioners in their software development. However, practitioners suffer from the lack of knowledge on software architectures and are not aware of potential benefits of specifying software architectures such as analysis of software architectures for quality properties. We attribute this to that practitioners mostly use Unified Modeling Language (UML) that is not powerful enough for understanding what one can do with software architectures.

4.1 Lack of Knowledge on Software Architectures

Software architecture is highly important for practitioners due to the nature of the software systems that practitioners develop. Despite that, very few practitioners know exactly what software architecture is. In their understanding, software architecture is just a design method for specifying the structure of software systems. However, very few are aware that software architecture is also a method for analysing the high level design of software systems for quality issues, e.g., safety, performance, and reliability. Indeed, specifying software architectures requires a considerable amount of effort; so, one should not use them just for communication but also for determining design defects.

Many practitioners have not ever used ADLs, sticking on UML. It is indeed the ADLs through which practitioners can perform further operations on their software architectures. There are various ADLs offered for different domains such as embedded systems, distributed systems, and multi-agent systems. Each ADL may also have different features, such as automatic implementation code generation, analysis for quality properties, simulation of system behaviours, etc. For instance, Wright (Allen and Garlan, 1997) is one of the most inspiring ADLs, which allows designers to specify software architectures in terms of components and connectors and then formally analyse their behaviours. There are also ADLs such as SOFA (Plasil and Visnovsky, 2002) that also focus on generating implementation code.

4.2 UML as the Most Popular Modeling Language

As already mentioned, UML is the top choice of most practitioners for specifying software architectures. Apparently, this is due to UML found easy to learn and use. Indeed, UML has been taught in many universities for teaching software design and architecture. Moreover, unlike ADLs, UML offers a comprehensive set of visual diagram types, making the design task simpler and more understandable.

While UML reduces the learning curve, its focus is limited with architectural specification. Issues such as analysing software architecture specifications for design errors or quality properties (e.g., performance and reliability) are ignored in UML. According to the survey results, many practitioners consider the analysis of software architectures as a crucial aspect of software design. Indeed, they agree that a modelling language that also enables analysis would be much more preferable. While practitioners can analyse their architectures using ADLs, ADLs are not found as familiar as UML due to their complex algebraic notations.

4.3 The Need for a Language that has Low Learning Curve and Supports Analysis

So, given the discussion above, if there was an architecture modeling language that supports *(i)* low learning curve, *(ii)* visual notation set, and *(iii)* architectural analysis, practitioners would be very likely to use that language. To this end, we currently work on a new architecture description language called XCD (Ozkaya and Kloukinas, 2014) that supports the above mentioned features in one place. Unlike the standard UML notation, XCD considers components and connectors (i.e., interaction protocols) as first-class architectural elements. To reduce the learning curve, unlike algebraic ADLs that are based on complex process algebras, XCD is based on the well-known Design-by-Contract approach (Meyer, 1992) and enables the precise and contractual specification of components and interaction protocols. XCD supports the analysis of software architectures in terms of its precise translation into the SPIN's ProMeLa formal verification language (Holzmann, 2004), which is automated via a tool that we developed[2]. So, unlike UML, in XCD designers can translate their contractual XCD architectures into ProMeLa models automatically and use the SPIN model checker to analyse their architectures for a number of properties, including deadlock, race-condition, wrong use of component services, and completeness. Now, we have been improving XCD with a visual notation set and a drawing editor, through which practitioners will be able to specify their XCD architectures visually and analyse them.

[2]XCD's website: https://sites.google.com/site/ozkayamert1/home/xcd

5 RELATED WORK

There have been many surveys conducted on software architectures, e.g., (Woods and Hilliard, 2005; Medvidovic and Taylor, 2000; Vestal, 1993; Fuxman, 2000). They shed light into what ADLs are basically for and what practitioners can do with the ADLs that may differ depending on the scope and the domain of each ADL. These surveys are extremely helpful in getting introduced with ADLs and their useful features such as formal analysis for quality properties and automatic code generation. However, it is not possible to understand from these surveys whether the current set of ADLs are successful in meeting the needs of practitioners. Indeed, what practitioners care most about may not be any of the features discussed in these surveys. Moreover, ADLs are just one of aspect of software architectures. There are other architecture modelling approaches, such as UML, which are outside the scope of these surveys. So, we are more interested in understanding whether practitioners are satisfied with the existing architecture modeling approaches and what features they look for in an architecture modeling approach.

Recently, another survey has been conducted by Malavolta et al. (Malavolta et al., 2012). Unlike the aforementioned surveys, Malavolta et al.'s survey focuses on determining what practitioners expect from ADLs. Malavolta et al.'s survey sheds light into very interesting results. For instance, they determined that practitioners are not so interested in using ADLs, instead using UML. This is due to the steep learning curve required by the ADLs. Moreover, Malavolta et al. also determines that formal analysis is one of the main reasons of practitioners in specifying software architectures. However, Malavolta et al.'s survey does not help in understanding practitioners' view of software architectures, and their level of knowledge and experience. Therefore, while one can understand why practitioners stay away from ADLs, it may not be so easy to understand what software architecture actually is to the practitioners.

6 ASSUMPTIONS AND VALIDITY

6.1 Sampling

In this survey study, we chose participants from both industry and academia so as to reduce any biases towards our work. Indeed, software architecture is not only used by industrial experts but also academics who can teach software architecture or do research on it. We used convenience sampling in choosing the participants for our survey. We determined the industrial experts from a number of IT companies developing software systems in various fields such as aviation, finance, telecommunication, and automation, etc. Our choice of the IT companies were random without paying any special attention to their profile or the industry field in which they do their business. Moreover, we determined the universities and the academics that we have contracted randomly too. The only constraint on our choice of academics is for focussing on those who have teaching/research background on software engineering.

6.2 Survey Technique

We conducted our survey as a questionnaire, thus expecting participants to choose one (or multiple ones) of the pre-defined answers for a set of questions. To get practitioners' views in the most complete way possible, we have also considered having interviews with the participants on the survey questions. Unfortunately, most practitioners were not willing to separate their times for that. Therefore, we instead decided to include the "other" option in some questions in which we believe participants may like to give their own particular answers. Then, we applied the coding technique to predict the most closest predefined answers to the participants' own answers.

6.3 Threats To External Validity

Having analysed our survey results, we reached four different findings, discussed in Section 3. These are *(i)* practitioners can facilitate the development of their systems through software architectures, *(ii)* the analysis of software architectures is a new concept to participants, *(iii)* architecture description languages (ADLs) are shown lack of interest by the participants, and *(iv)* UML is popular with its low learning curve and visual notations, while still suffering from the lack of support for architectural analysis. However, the above findings are based on our survey among 50 participants who have been chosen from the companies located in Turkey. Therefore, the findings herein may not necessarily be generalized to the entire community of software engineering.

7 CONCLUSION

Software architecture has been one of the most crucial topics in software engineering since the nineties. Various techniques have been developed to support the specification of software architectures, such as Unified Modelling Language (UML) and architecture description languages (ADLs). However, it always remains ambiguous what practitioners know about software architecture and how they use it in their software design. To understand this, we conducted a survey among 50 different practitioners, including both industrial experts and academics. We asked 20 questions and analysed their answers by evaluating each question individually and also the relations between the questions. According to our analysis results, software architecture has high potential in facilitating the software development of practitioners. However, despite that, practitioners are not familiar with software architectures. They use software architectures for specifying the structure of systems, but ignoring other aspects such as the analysis of software architectures for design errors and quality properties. This is because practitioners restrict their scope with UML, which enables the visual – i.e.,friendly – specification of software architectures but does not provide direct support for their analysis.

REFERENCES

Allen, R. and Garlan, D. (1997). A formal basis for architectural connection. *ACM Trans. Softw. Eng. Methodol.*, 6(3):213–249.

Bergstra, J. A. (2001). *Handbook of Process Algebra*. Elsevier Science Inc., New York, NY, USA.

Canal, C., Pimentel, E., and Troya, J. M. (1999). Specification and refinement of dynamic software architectures. In Donohoe, P., editor, *WICSA*, volume 140 of *IFIP Conference Proceedings*, pages 107–126. Kluwer.

Clements, P. C. (1996). A survey of architecture description languages. In *Proceedings of the 8th International Workshop on Software Specification and Design*, IWSSD '96, pages 16–, Washington, DC, USA. IEEE Computer Society.

Clements, P. C., Garlan, D., Little, R., Nord, R. L., and Stafford, J. A. (2003). Documenting software architectures: Views and beyond. In Clarke, L. A., Dillon, L., and Tichy, W. F., editors, *ICSE*, pages 740–741. IEEE Computer Society.

Feiler, P. H., Gluch, D. P., and Hudak, J. J. (2006). The Architecture Analysis & Design Language (AADL): An

Introduction. Technical report, Software Engineering Institute.

Fuxman, A. D. (2000). A survey of architecture description languages. Technical Report CSRG-407, Department of Computer Science,University of Toronto, Toronto, Ontario, Canada M5S 3G4.

Garlan, D. and Shaw, M. (1994). An introduction to software architecture. Technical report, Pittsburgh, PA, USA.

Hoare, C. A. R. (1978). Communicating sequential processes. *Commun. ACM*, 21(8):666–677.

Holzmann, G. J. (2004). *The SPIN Model Checker - primer and reference manual*. Addison-Wesley.

Ivers, J., Clements, P., Garlan, D., Nord, R., Schmerl, B., and Silva, J. R. O. (2004). Documenting component and connector views with UML 2.0. Technical Report CMU/SEI-2004-TR-008, Software Engineering Institute (Carnegie Mellon University).

Luckham, D. C. (1996). Rapide: A language and toolset for simulation of distributed systems by partial orderings of events. Technical report, Stanford University, Stanford, CA, USA.

Magee, J. and Kramer, J. (1996). Dynamic structure in software architectures. In *SIGSOFT FSE*, pages 3–14.

Magee, J., Kramer, J., and Giannakopoulou, D. (1997). Analysing the behaviour of distributed software architectures: a case study. In *FTDCS*, pages 240–247. IEEE Computer Society.

Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., and Tang, A. (2012). What industry needs from architectural languages: A survey. *IEEE Transactions on Software Engineering*, 99.

Medvidovic, N. and Taylor, R. N. (2000). A classification and comparison framework for software architecture description languages. *IEEE Trans. Software Eng.*, 26(1):70–93.

Meyer, B. (1992). Applying "Design by Contract". *IEEE Computer*, 25(10):40–51.

Milner, R., Parrow, J., and Walker, D. (1992). A calculus of mobile processes, i. *Inf. Comput.*, 100(1):1–40.

Ozkaya, M. and Kloukinas, C. (2014). Design-by-contract for reusable components and realizable architectures. In Seinturier, L., de Almeida, E. S., and Carlson, J., editors, *CBSE'14, Proceedings of the 17th International ACM SIGSOFT Symposium on Component-Based Software Engineering (part of CompArch 2014), Marcq-en-Baroeul, Lille, France, June 30 - July 4, 2014*, pages 129–138. ACM.

Perry, D. E. and Wolf, A. L. (1992). Foundations for the study of software architecture. *SIGSOFT Softw. Eng. Notes*, 17(4):40–52.

Plasil, F. and Visnovsky, S. (2002). Behavior protocols for software components. *IEEE Trans. Software Eng.*, 28(11):1056–1076.

Reussner, R., Poernomo, I., and Schmidt, H. (2003). Reasoning about Software Architectures with Contractually Specified Components. In Cechich, A., Piattini, M., and Vallecillo, A., editors, *Component-Based Software Quality*, volume 2693 of *Lecture Notes in Computer Science*, page 287âĂŞ325. Springer Berlin Heidelberg.

Rumbaugh, J. E., Jacobson, I., and Booch, G. (1999). *The unified modeling language reference manual*. Addison-Wesley-Longman.

Shaw, M., DeLine, R., Klein, D. V., Ross, T. L., Young, D. M., and Zelesnik, G. (1995). Abstractions for software architecture and tools to support them. *IEEE Trans. Software Eng.*, 21(4):314–335.

Spivey, J. M. (1992). *Z Notation - a reference manual (2. ed.)*. Prentice Hall International Series in Computer Science. Prentice Hall.

Taylor, R. N., Medvidovic, N., Anderson, K. M., Jr., E. J. W., Robbins, J. E., Nies, K. A., Oreizy, P., and Dubrow, D. L. (1996). A component- and message-based architectural style for gui software. *IEEE Trans. Software Eng.*, 22(6):390–406.

Vestal, S. (1993). A cursory overview and comparison of four architecture description languages. Technical report, Honeywell Technology Center.

Woods, E. and Hilliard, R. (2005). Architecture description languages in practice session report. In *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture*, WICSA '05, pages 243–246, Washington, DC, USA. IEEE Computer Society.

APPENDIX

In the Appendix, you will find a table for each survey section, in which the percentages of the responses are shown for the questions of that section. Note that we have not included the questions of the first section (i.e., Personal Information introduced in Section 2.2.1) as the responses for those questions are not discussed in Section 3 for finding an answer to the research questions. Note also that for some questions, the percentages are not integer values (e.g., 42.8). In such cases, the nearest integer values are given (e.g., 43 for 42.8).

Table 1: Survey Results for Section "The types of projects that are conducted by practitioners".

Q1	Do you build critical systems (where people may die/get injured when they fail or the system gets damaged at cost or the business loses great amounts of money on failures)?
	Answers: 40% of the participants responded "No", while 60% responded "Yes".
Q2	Do you build large systems with more than one development team?
	Answers: 43% of the participants responded "No", while 57% of the participants responded "Yes".
Q3	Do you build long-lived systems that go through years of maintenance and adaptation?
	Answers: 28% of the participants responded "No", while 72% of the participants responded "Yes".
Q4	Do you build families of systems/products such as client-server systems?
	Answers: 30% of the participants responded "No", while 70% of the participants responded "Yes".

Table 2: Survey Results for Section "Practitioners' understanding of software architectures".

Q1	Which of the following role do you think suits best for software architects?
	Answers: 34% of the participants responded "A software architect designs the software system (its structure) to be built." 36% of the participants responded "A software architect decomposes a system into components, and then analyse these components along with their interaction for quality properties."; 15% of the participants responded "A software architect analyses the problem domain to understand the problem and make decisions."; 6% of the participants responded "A software architect decides on the requirements of a software system."; and 9% of the participants responded "other".
Q2	Have you taken or given any course on software architectures before?
	Answers: 45% of the participants responded "No", while 55% of the participants responded "Yes".
Q3	Have you attended any seminar/conference on software architectures before?
	Answers: 62% of the participants responded "No", while 38% of the participants responded "Yes".
Q4	Have you done any research on software architectures?
	Answers:51% of the participants responded "No", while 49% of the participants responded "Yes".

Table 3: Survey Results for Section "The level of interest shown by practitioners towards software architectures".

Q1	In the projects you are involved in, do you always consider the architectural aspects of software systems?
	Answers: 79% of the participants responded "Yes", while 21% of the participants responded "No".
Q2	Are you aware of any architecture description languages?
	Answers: 43% of the participants responded "Yes", while 57% of the participants responded "No".
Q3	At which stage of software development do you prefer to check the correctness of your software system?
	Answers: 55% of the participants responded "Pre-coding stage (while designing the system)"; 26% of the participants responded "Coding stage (while coding the system)"; and, 19% of the participants responded "Post-coding testing stage (after developing the system)".
Q4	Which strategy do you follow in implementing your software system?
	Answers: 57% of the participants responded "Firstly design the system and then implement the system according to its design."; 2% of the participants responded "Directly implement the system without designing."; 26% of the participants responded "Firstly design the system, and then use techniques/tools to automatically generate code from the design document."; and 15% of the participants responded "other".
Q5	Did you know that architecture description languages enable the analysis of software architectures for design errors (e.g., incompatible components and deadlock) and quality properties (e.g., performance, reliability, and security)?
	Answers: 53% of the participants responded "No", while 47% of the participants responded "Yes".
Q6	Did you know that architecture description languages enable the automatic generation of implementation code from software architectures?
	Answers: 45% of the participants responded "No", while 55% of the participants responded "Yes".

Table 4: Survey Results for Section "Software architecture specification techniques used by practitioners".

Q1	Which languages/tools do you use for specifying software architectures?
	Answers: 89% of the participants responded "UML"; 44% of the participants responded "Office tools, such as Microsoft Word and OpenOffice"; 13% of the participants responded "Architecture Description Languages"; and, 15% of the participants responded "other".
Q2	What do you see as their advantages/useful features?
	Answers: 57% of the participants responded "Easy to learn and use"; 28% of the participants responded "Visual notations"; 28% of the participants responded "Automatic analysis for quality properties, such as reliability and performance"; 22% of the participants responded "Automatic analysis for design errors"; 20% of the participants responded "Automatic implementation code generation".
Q3	What do you see as their disadvantages/missing features?
	Answers: 45% of the participants responded "Difficult to use and learn"; 14% of the participants responded "Textual notations"; 62% of the participants responded "Lack of support for analysis"; 45% of the participants responded "Lack of support for automatic code generation"; and 11% of the participants responded "other".